REPORT DOCUMENTATION PA

AD-A274 694

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/94	Scientific Paper

4. TITLE AND SUBTITLE
PDEF: A Standard File Format for Data Interchange

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael McDonnell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Topographic Engineering Center
ATTN: CETEC-PAO
7701 Telegraph Road
Alexandria, VA 22310-3864

8. PERFORMING ORGANIZATION REPORT NUMBER
R-203

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Introduction

The Protean Data Exchange Format (PDEF) is a set of computer programs which can be used to transform data between different systems which do not understand each other's formats. The problem of data transformation is not restricted to any particular discipline, but those of us concerned with digital terrain data have felt this problem acutely as we have found that different Geographic Information Systems (GIS) cannot use each other's data. Although PDEF was written to alleviate this particular problem, its uses are more general than the specific problem for which it was originally written. PDEF was designed primarily to be easy to use since reformatting decisions must be made by people. This report will examine both the format and the uses of PDEF.

14. SUBJECT TERMS
Data Exchange, Data Transformation, Geographic Info Systems, Reformatting

15. NUMBER OF PAGES
10

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	

PDEF: A STANDARD FILE FORMAT FOR DATA INTERCHANGE

Michael McDonnell
U.S. Army Topographic Engineering Center
Fort Belvoir, VA 22060-5546

Introduction

The Protean Data Exchange Format (PDEF) is a set of computer programs which can be used to transform data between different systems which do not understand each other's formats. The problem of data transformation is not restricted to any particular discipline, but those of us concerned with digital terrain data have felt this problem acutely as we have found that different Geographic Information Systems (GIS) cannot use each other's data. Although PDEF was written to alleviate this particular problem, its uses are more general than the specific problem for which it was originally written. PDEF was designed primarily to be easy to use since reformatting decisions must be made by people. This report will examine both the format and the uses of PDEF.

This report is intended for both programmers and non-programmers who need to understand data formats and data format transform techniques. Programmers can find sufficient detail in the appendixes to implement PDEF. Programming examples are given there for various data types, such as raster, quadtree, and vector data. Non-programmers will find guidance in the body of the report on how to express their data-reformatting decisions in PDEF so that PDEF will reformat the data properly.

To understand PDEF, we need a brief review of data file formats. A *file* is a separate data entity on a disk or tape. It has its own name on a disk and is separated from other files on a tape by an end-of-file mark. Some files are further subdivided into records and fields. PDEF does not use any subdivisions finer than files. There are no records or prescribed fields defined in PDEF.

Data is commonly stored and transferred from one computer to another by using conventions in the formatting of the file, or files, containing data. It is common to have a *header* as the first part of a file. The header contains data about the file as a whole, such as the name of the data set, and also contains format information to help in reading the rest of the file. Headers typically contain a mixture of printable and binary data and are difficult to parse (i.e. understand) without a manual. There are no headers in PDEF. The rest of this introduction gives the rationale for the design of PDEF.

Designing a new data format (and implementing tools such as parsers to allow people to work with it) is a large undertaking and should only be done if there is a strong need. Is there, when, a good reason for defining PDEF? One reason is the influence of old formats and old languages such as FORTRAN. FORTRAN has dominated the design of file formats to date and causes many of the problems users have when attempting to understand and work with a current format.

FORTRAN cannot allocate data dynamically. This lack of dynamic allocation forces data file formats to have fixed field lengths, which causes some problems. For example, if a user wants to name a file using a title that is 20 letters long but the data file has only set aside 15 spaces for the name, then the desired filename must be shortened to a length of 15 letters or less. Carried to extremes, this leads to names for functions and variables that are restricted

to a few upper-case ASCII characters and therefore have almost no semantic content. What does a function named SAXPY or QRTPE do? There is no way to tell without a manual. Similarly, and more to the point of this discussion of data formats, what data does field FTLLP contain? Again, there is no way to tell.

A problem encountered when dealing with many current data formats is a waste of space on data transfer media. If fixed-length fields have to be made large enough to contain the largest expected data element, then for an average data element there will be unused space that must be transferred on the tape anyway. Attempts to overcome this limitation, such as the ISO 8211 data transfer standard,[1] have a daunting complexity. The author has recently written a parser for ISO 8211 and it was a difficult task.

Another problem with most current formats is that they were designed by committees and therefore have a lack of conceptual integrity and an unnecessary complexity which is characteristic of such works. As an example, the proposed spatial data transfer specification which has been published by the Digital Cartographic Data Standards Task Force (DCDSTF) has a specification that is over 120 pages long.[2] Such specifications as DCDSTF are unworkably complex and will have to be changed later, leading to problems with versions of the "standard." In contrast, PDEF is simple. This simplicity is mostly a result of separating syntax from semantics, as will be illustrated later. Simplicity and human-readability were the guiding precepts in its design. However, simplicity must not be sacrificed to usability or usefulness. PDEF proves that a format can be both simple and useful.

A GENERAL DESCRIPTION OF PDEF

In PDEF a single data set is typically stored in several separate files, with (mostly) meta-data in those files that are readable by people. Meta-data is data about data. It includes such information as where the data is found, how many bits there are per data element, and whether data is to be read as a string of characters, as an integer, as an array of 32-bit floating-point numbers, etc. Although most current file formats contain such information in a file "header," PDEF has no file headers. In PDEF, a separate, human-readable file contains the meta-data typically found in headers. The bulk of the data is then placed in another file or set of files that contain nothing but binary data. Binary data files have the following characteristics:

---

[1] *Information Processing - Specification for a data descriptive file for information interchange*, International Organization for Standardization publication ISO 8211-1985(E), 15 Dec 1985.

[2] Digital Cartographic Data Standards Task Force, "Draft Proposed Standard for Digital Cartographic Data" *The American Cartographer*, vol. 15, p. 21.

- no headers
- no trailers
- no field padding
- no field separators (at least none required)

in short, no wasted space. With the exception of a defined field separator, these characteristics are also true of meta-data files, which I will call *information files* from here on.

The issue of human readability needs to be discussed. Why is it desirable to have data files be readable by people? Are there penalties to be paid in computer efficiency or in storage usage if human-readable information files are used?

A benefit of human-readable information files is that a manual is not needed to understand something about a data set. It frequently happens that a data set is presented to a potential user without any accompanying documentation. Therefore, understanding something about the data without referring to auxiliary documentation is often useful, and can even be crucial, since you can't read the data unless you know its formats.

Another benefit of human-readable information files is that it is easier to write a parser for these files than if nonprintable data has to be dealt with. Appendix C illustrates parsing of a SPOT[3] data file using the fixed-field data that comes on the SPOT tape. Appendixes B, D, and E illustrate parsing of the same file using a PDEF information file. You can see that the parsing is much more understandable when using PDEF. As illustrated in appendix D, It is also possible to use the parsing tools *lex*[4] and *yacc*,[5] which are of great help in writing parsers. While lex and yacc were created on Unix systems, they are now available under all major operating systems such as MSDOS, OS/2, and VMS.

Human-readable files do take more storage on disk or tape than binary files of the same data. For this reason, PDEF defines two types of files. The human-readable file is only used for data that must be read to understand the data set as a whole. Its inefficient storage is not a problem since binary data files are much larger than human-readable files for the large data sets that PDEF was created to manipulate.

Having mentioned the possible usefulness of PDEF, the PDEF file formats will now be described as will some of the software tools that manipulate them. The body of this report describes the abstract characteristics of PDEF, which are simple, but file formats rely on a consideration of detail. This detail is given in the appendixes which provide examples of possible uses of PDEF.

---

[3] *Format for the SPOT Image Corporation Computer Compatible Tapes*, SPOT Image Corporation, Aug 1986.

[4] M. E. Lesk and E. Schmidt, "Lex - A Lexical Analyzer generator" in *Unix Time-Sharing System: Unix Progammer's Manual, Vol. 2B 7th edition*, AT&T Bell Laboratories, 1979.

[5] S. C. Johnson, "Yacc: yet another compiler-compiler" in *Unix Time-Sharing System: Unix Progammer's Manual, Vol. 2B 7th edition*, AT&T Bell Laboratories, 1979.

A large data set in PDEF consists of at least two separate files. One of these files is an *information file*, which contains general information about the data such as file offsets and how the data are to be parsed. Appendix B contains a PDEF information file for SPOT satellite data. There is usually only one information file for a given data set. Besides the information file, there is usually at least one *binary data file*. The information file format will be discussed first, followed by a discussion of the binary data file format.

THE INFORMATION FILE

Information files consist only of printable ASCII characters. In the ASCII numeric sequence, printable ASCII characters include characters ' '(space) through '~' (tilde) inclusive and also include the hexadecimal characters 0A (newline) for line breaks and 09 (tab) for spacing. This is all in accordance with the C programming language practice of considering "white space" characters to be among the printable ASCII set, where white space characters are defined as space, tab, and newline. No other characters are allowed in information files.

An information file without nonprintable characters can be easily viewed without bombarding a terminal or workstation with what may be in-band control information, thus putting it into undesirable states. No special programs are needed to view the information. Any program that writes text data onto the screen is usable for viewing information files, and ordinary text manipulation programs can be used to create or modify information files.

Information files have two reserved characters, the pound sign '#' and the colon ':'. The pound sign character '#' indicates a non-parsable comment. Any characters on a line from the first occurrence of a '#' until the end of the line (i.e. until a newline) are not read by the information file parser. The other reserved character is the colon ':' which is used to separate a *key* from the data associated with the key. Appendix A shows an example information file that has been rather strangely formatted to show some of the possible uses of the '#' and ':' characters.

The "key" is purposely not called a "keyword" because a "key" is a phrase that may contain many words. Keys should be designed to be very descriptive. A poor key would be cryptic such as "redoff," while an equivalent good key would be "file offset to beginning of red image." Keys must begin a line. This means that either they must appear at the very beginning of the information file or they must always follow a newline. Keys are separated from the data to which they refer by a colon ':'. Leading and trailing spaces or tabs in a key are ignored by the parser. See the comments in the information file in Appendix A for further details.

Appendix A also shows how a single key may refer to a data structure rather than a single data item. There the structure is a colormap which consists of a repeated sequence of [pixel red green blue] values. Data structures can be defined in a PDEF information file too. For example, there may be entries like this:

```
#
# establish how we encode a colormap
#
colormap sequence: red green blue pixel
color maximum value: 65536 # for the X Window System
pixel maximum value: 255
colormap:
        3456   12345 8976   0
        12345  8974  3458   1
        # and so on ...
```

Other types of composite data, such as matrices or coordinate tuples, may be handled similarly. Here is a possible representation of Quam's block storage of raster data,[6] such as is used for ARC Digitized Raster Graphics, a product of the U.S. Defense Mapping Agency.

```
Data type: raster              # also vector, quadtree, etc.
Storage format: blocked              # could be RGB, band interleaved, etc.
Block size: 128
        # and so on ...
```

An example of how quadtrees may be encoded will be shown later. Appendix F presents a design for storing vector and polygon data.

BINARY DATA FILES

It may be that a data set encoded in PDEF does not contain any binary data files whatever; all of the data being placed in the information file. This is only reasonable, though, for small data sets. For large data sets, one or more binary data files should be used in addition to the information file.

Binary data files contain data in which the largest guaranteed unit of reference is the 8-bit byte. The information file tells the user how to interpret these data bytes. Because of the vagaries of byte ordering on different computers, the information file may specify how to assemble larger data units from bytes. For example, data bytes may be read in the order 1 2 3 4, but a 32-bit integer formed from these bytes may have to be written in the order 2 1 4 3.

Data ordering is a significant problem. The author uses the conventions described in Sun Microsystem's eXternal Data Representation (XDR) standard for representing more complex data types. XDR is explained in the document RFC1014 which may be gotten through the Internet by ftp from nic.sri.com or by request from Sun Microsystems, Inc. Data type encoding is, however, not enforced by PDEF and so will not be discussed here further.

---

[6] L.H. Quam, "A Storage Representation for Efficient Access to Large Multidimensional Arrays", *Proceedings DARPA Image Understanding Workshop*, 104-111, 1980.

Appendix E shows that a separate information file may be used to work with some data format *without* reformatting it into an intermediate binary format. The programs in Appendix E parse a SPOT file in its distributed format; headers in the SPOT data are just ignored. This technique allows a common set of parsing and data manipulation programs to work on various types of data without reformatting the data file itself. This is a significant advantage for large data sets where reformatting the data would take much computation and use a large amount of storage.

USE OF PDEF FOR DATA REFORMATTING

As mentioned in the introduction, nothing except valid data should be stored in binary data files. If applicable, one may, of course, specify fixed-length fields, padding, headers, and all the other apparatus found in various file structures. This flexibility has a use in that it is by this means that data can be exchanged from one format to another. It is this problem, data reformatting and exchange, that inspired work on PDEF (and is the source of the name *protean*).

The combinatorics of data reformatting mandate a common intermediate format. The following table shows how the number of parsing programs needed is affected by the presence of an intermediate file format.

Table 1. Number of Parsing Programs Needed

number of formats	parsing programs needed without PDEF	parsing programs needed with PDEF
8	56	16
9	72	18
10	90	20
100	9900	200

The relevant mathematics are that without a common file format one needs n(n - 1) reformatting programs and with a common file format only 2n reformatting programs. Thus, the first problem is of order n squared while the second problem is of order n.

The table doesn't tell the whole story, though. The programs that have to be written fall into two equal-sized groups, which is where the factor of 2 in 2n comes from. Figure 1 shows the situation. The programs that convert some other format into PDEF have a common back end in that they all feed into PDEF. Similarly, the programs that convert from PDEF to another format have a common front end that reads the PDEF file. When these commonalities are considered, the problem simplifies further to essentially n programs instead of 2n programs. There is no doubt that an intermediate file format is needed if data reformatting among many formats needs to be done.

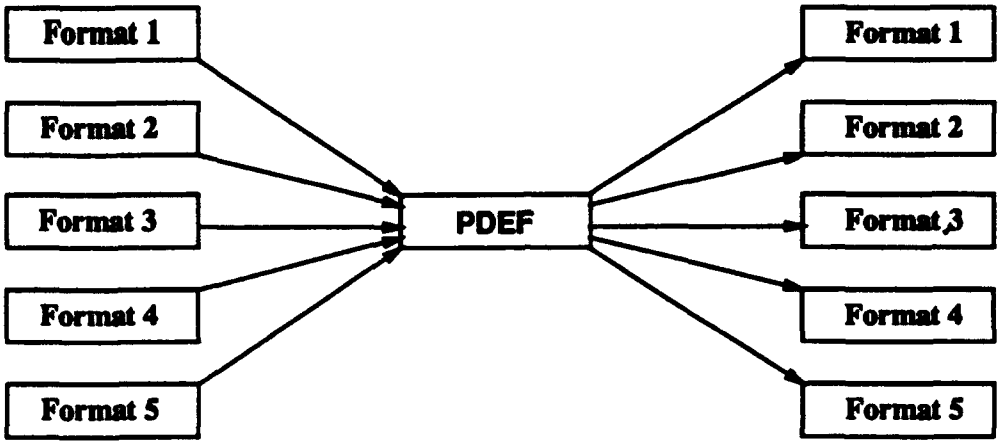

Figure 1: Diagram of interchange among five data formats using PDEF as an intermediate format. All data formats may be reformatted to PDEF and then PDEF may be reformatted into any other format. The arrows represent programs that perform data reformatting and the rectangles represent the formats.

PDEF AS A PRIMARY DATA FORMAT

Besides serving as a means of reformatting data, PDEF can itself be a primary data format. This means that devices such as image scanners and telemetry systems can encode their data in PDEF for transmission. The readability of PDEF information files then allows the operator to quickly check data content. Software systems such as a GIS can also be based on PDEF, thereby allowing easier conversion of data to and from foreign formats. The U. S. Army Engineer Topographic Laboratories (ETL) will use PDEF as the format for data generated by the Terrain Information Extraction System (TIES), which is a developmental system allowing Army units to extract terrain data from digital photography in the field.

TYPES OF DATA THAT MAY BE HANDLED

Current experience with PDEF has been only with regard to raster data files. In order to be useful as a general data exchange file (i.e. in order to be truly *protean*), PDEF has to be able to handle other data types and structures as well. An example relevant to ETL is GIS data, which includes vectors (with associated attributes) and quadtrees as well as gridded (raster) data. Below is an example of a quadtree structure defined in PDEF. Although the following is not the only way to encode a quadtree in PDEF, it is an example meant to indicate that PDEF is capable of doing this.

One way to store a quadtree on a disk or tape is to define a traversal order of the tree and then to linearize the tree to a file by traversing it. To rebuild the tree from the file, just build it

in the same order when the file is read. Here is a section of an information file dealing with quadtrees:

```
Traversal order:                          preorder
Quadtree node data order:                 attribute NW NE SE SW
Attribute:                                generic pointer
NW:                                       boolean
NE:                                       boolean
SE:                                       boolean
SW:                                       boolean
#
# binary data file is just quadtrees, so offset is zero
#
Offset of root quadtree in data file:     0
```

Given this information, a parser program can go through the binary data file and add in the nodes for which the boolean attributes of its father node are TRUE, meaning that there is a node under this quadrant. This simple scheme defines leaf nodes as having all four quadrants FALSE and with an attribute assigned to its area. Note that storage can be saved by encoding the four leaf nodes in a single byte since they only need one bit each to perform their function. Data descriptions, such as "boolean" or "generic pointer," can be further described by other entries in the information file. Other needed data would certainly include geographic coordinates of the corners of the region encoded in the quadtree. Individual quadtree nodes need not carry their coordinates along with them since these are implicit in the tree.

Another data type of great interest to the GIS community is vector data. A realistic design of a vector data format for PDEF is too big to be given in the body of this report; however, Appendix F contains an example design based on a United States Geological Survey format for vector and polygon data, such as is used in a GIS. All essential information such as vector ordering is preserved.

GENERAL DISCUSSION; SYNTAX AND SEMANTICS

PDEF defines the syntax of a data exchange file format and not the semantics of such a format. PDEF does not address some of the most difficult problems associated with reformatting data, such as forcing a match between data fields that are not strictly conformable. If a name field in one format has 30 characters allocated to it and another only allows 10 characters, how is a conversion to be made? Similarly, if needed data in some format is simply not available in a precursor format, what defaults should be used to fill in the blanks in the output format? Should they be filled at all? Problems such as these are matters of policy and are therefore beyond the scope of PDEF because it is only a file formatting vehicle. However, PDEF makes it easier to address these issues of reformatting policy.

For one thing, having a protean and human-readable format for information storage and exchange means that those people charged with deciding the form of data storage or interchange can encode their decisions directly in the information file that is to be parsed by a computer. This makes the data formatting readily and directly reviewable. There is no danger of a mistranslation between what the computer must read and what people can read since information files can be read by both people and computers. Using the information file for this purpose prevents such mismatches. It is best to avoid the production of auxiliary documents detailing formatting decisions since the PDEF information file and the auxiliary document may disagree.

There are some concerns that need to be addressed which come from having a separate file that includes the parsing information for binary data files. These concerns are as follows:

- The information file may be separated from the data files to which it refers.
- The information file is both too easy and too difficult to change. It is too easy to change because it is a text file and can therefore be modified by a text editor so that it no longer has accurate data. It is too difficult to change because a program may modify the binary data without modifying the information file.

In general, a concern exists that there is too much decoupling between the information file and the data it describes.

Although the information file may indeed be separated from the file it describes, many current data formats make use of separate files and this does not seem to be a significant problem. SPOT and ADRG data are both distributed as sets of files for a single coverage area. These files have a complex interrelationship that is much more difficult to work with than the simple scheme described here. Since no documentation exists which relates operational problems caused by these sets of files becoming dissociated, this is probably more of a theoretical than a practical problem.

Having the information file track the data file is a matter of convention that is not enforced by the format itself. A reasonable convention is to have information files be protected so that they are read-only for users and can only be operated on by privileged programs. Data files can be treated in the same way. It is then the responsibility of programs that modify the data to concurrently modify the information file. To be even more certain that information files and data files are consistent, a given set of information and data files should not be modifiable at all except under regulated circumstances. What is meant here is that if the data set is to be modified, it must be copied to a new data set and a new information file generated to describe it. This keeps a history of data processing information, which it is probably desirable to have anyway. Advances in data storage media have alleviated the problem of storage of large data sets, and data can in any case be overwritten, if this is needed.

Pascoe and Penny have recently critiqued the problem of producing a standard inter-

change format for GIS data.[7] Since they do not start from the assumption that there can be a separation of syntax from semantics for a data exchange format, they are led to the conclusion, repeatedly stated, that any such standard must be very complex. Indeed, if all the decisions about data semantics are considered, then the result is very complex. This report proposes that the format of interchange may itself be very simple and can help with the more difficult policy problems concerning data reformatting.

Data for an output record in one format may have to be derived from a number of files in some other format. This means that searching of several files must be done for these cases to generate a single output datum. Pascoe and Penny advocate reading all input data into a relational database management system (RDBMS) to distribute the data into a set of relations that can then be searched, as needed, for outputting a new format.[8] While this insulates the programmer from explicit searching for data, the redistribution of data may not be a frequent occurrence, and the apparatus of an RDBMS seems unnecessary. This is certainly true of raster data, which will be closely associated in any format and therefore does not have to be redistributed. Pascoe and Penny mention the large amount of computer resources needed when using an RDBMS;[9] file searching as needed should not impose as much of a burden. Experience with PDEF will show whether it is an efficient means of transferring information or whether an RDBMS is a needed adjunct. As mentioned above, much of the initial data may be retained in the original format and only parsed out as needed. The goal should be to do as little reformatting as possible.

SUMMARY AND CONCLUSIONS

PDEF provides a powerful, simple, and human-readable method of encoding data in a form that is easily parsable by automatic computer methods. Since parsers are written in *yacc*, they are expressed in a formal syntax grammar (Backus-Naur form) and are easy to write and modify. PDEF is superior in simplicity and power to other formats and allows efficient data storage. Owing to its simplicity, it is unlikely that future versions of PDEF need to be designed. This ensures that there will not be outdated versions of PDEF that must be accommodated in the future. PDEF does not have the disadvantages pointed out by Pascoe and Penny for other data exchange formats and can serve as a much more tractable and protean standard than current formats.

---

[7] R. T. Pascoe and J. P. Penney, "Construction of Interfaces for the Exchange of Geographic Data" *Int. J. Geographical Information Systems*, vol. 4, No. 2, 147-156, 1990.

[8] Ibid.

[9] Ibid.